```
Grammar_Elem:
    Kind:
        - Nonterminal
        - Terminal
     -> the kind of the elem

     std::string str
       -> the string representation

Grammar_Production:
    Grammar_Elem nonterminal;
      -> the nonterminal at the left hand side of the production
    std::vector<Grammar_Elem> elems;
      -> the list of elems at the right hand side of the production

Gramar:
    Grammar_Elem starting_nonterminal;
        -> starting nonterminal
    std::vector<Grammar_Elem>       elems;
        -> the elems
    std::vector<Grammar_Production> productions;
        -> the productions


    Grammar Grammar_read_from_file(std::string path)
        -> reads the grammar from a file with the syntax of
            grammar = starting_nonterminal productions.
            starting_nonterminal = "starting_nonterminal:" nonterminal ";".
            productions = "productions:" {production} ";".
            prodcution  = nonterminal ":" {(nonterminal | terminal) "," } ";".
            nonterminal = (letter | digit) { (letter | digit) }.
            digit  = "0" | ... | "9".
            letter = "a" | ... | "z" |
                     "A" | ... | "Z".
            terminal = "\"" (* any character except "*) "\"".

    bool Grammar_cfg_check(Grammar grammar)
        -> checks if the grammar is cfg

    Option<Grammar_Production> Grammar_get_production(Grammar grammar, Grammar_Elem nonter
minal, usize i = 0)
        -> Gets the i-th production of the nonterminal from the grammar if exists

Recursive_Descent_Parser:
    RDP_State state;
    usize i = 0;
        -> Position in sequence
    std::vector<RDP_Elem> working_stack;
    std::vector<RDP_Elem> input_stack;

    std::string sequence;
    Grammar grammar;
        -> Grammar used for parsing

    Recursive_Descent_Parser RDP_create(Grammar grammar, std::string sequence)
```

```
        -> Creates a parser

    void RDP_step(Recursive_Descent_Parser* rdp)
        -> Does one step

    void RDP_print(Recursive_Descent_Parser rdp)
        -> Prints the parser

    Recursive_Descent_Parser rdp_parse(Grammar grammar, std::string sequence);
        -> Parses the sequence using the specified grammar.
           Returns the parser. It is either in Error or Final state.

    Node rdp_create_tree(Recursive_Descent_Parser parser)
        -> Creates the node tree from the parser.
           The parser must be in his Final state.

Node:
    Grammar_Elem elem;
    std::vector<Node> children;

    void node_print(Node node, usize tab = 0)
        -> prints the node in a tree-like fashion.
```